Deep Optimization of VMM Live Upgrade

Shenming Lu

ByteDance

Agenda

- Background
- Overview of VMM Live Upgrade
- Downtime Breakdown
- Optimizations
- Achievements

Background

Background

VMM live upgrade:

- upgrade the VMM (QEMU & KVM) without interrupting VMs
- add security patches and new features

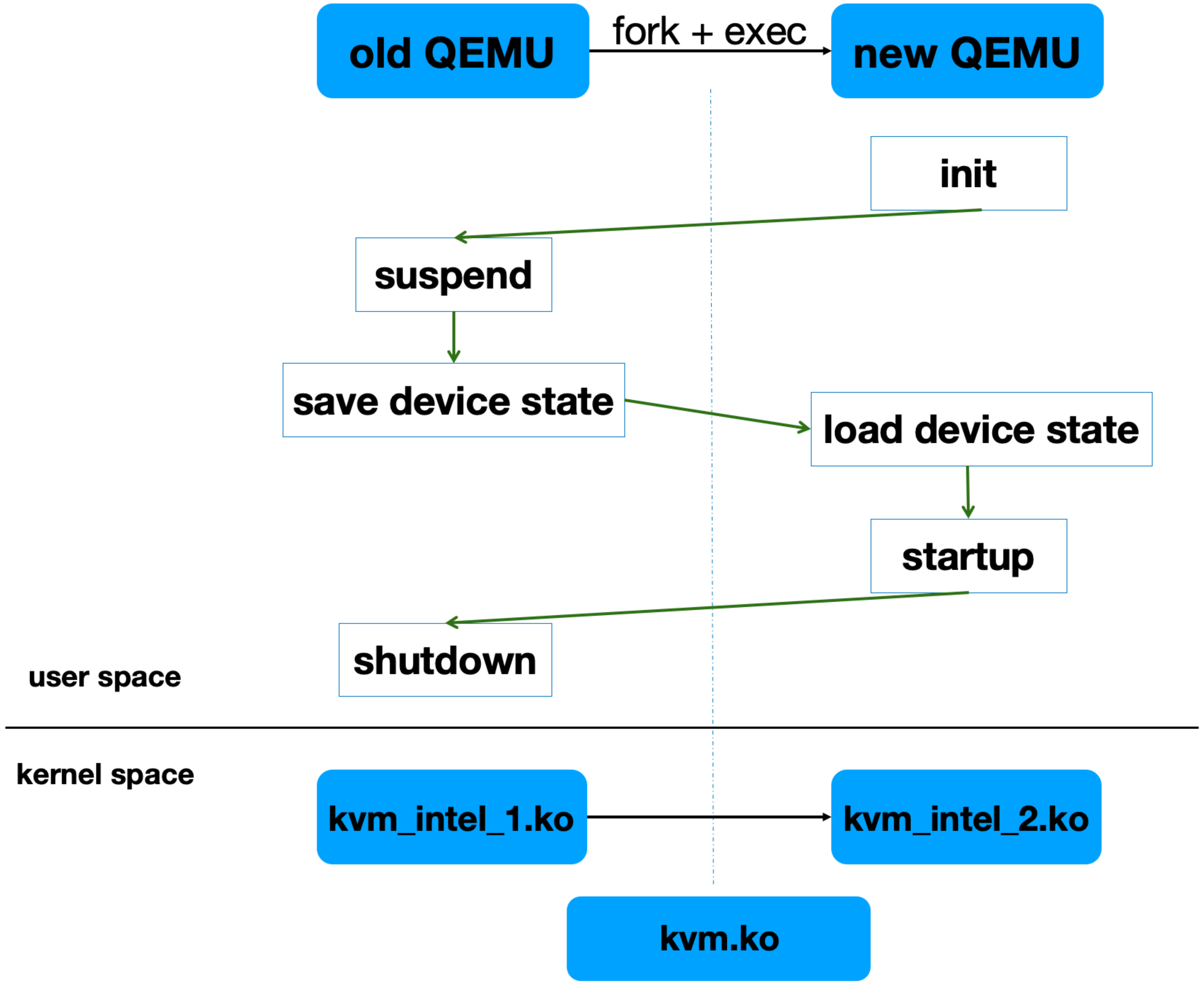
Background

Main issue:

- minimizing service downtime is still the major concern of cloud providers
- downtime for large VM can be as long as several seconds

Overview of VMM Live Upgrade

Overview of VMM Live Upgrade



- Use fork+exec to load the new QEMU binary

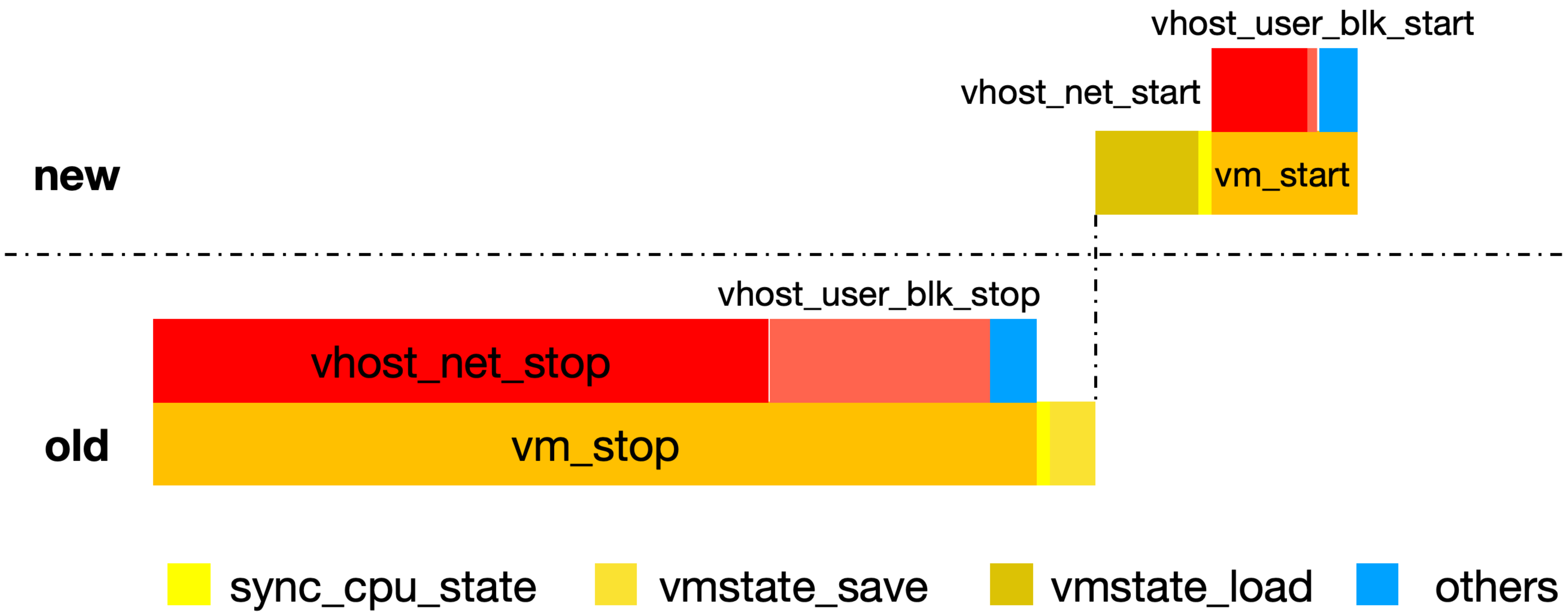
  - can inherit any fd from the old, including memfd…

- Use shared memory to sync and transfer device state between the old and new
- Divide the kvm module into multiple duplicated modules to also upgrade kvm

Downtime Breakdown

Downtime Breakdown

64 vCPUs, 256G memory, 1 multiqueue vhost-user-net device, 2 multiqueue vhost-user-blk devices



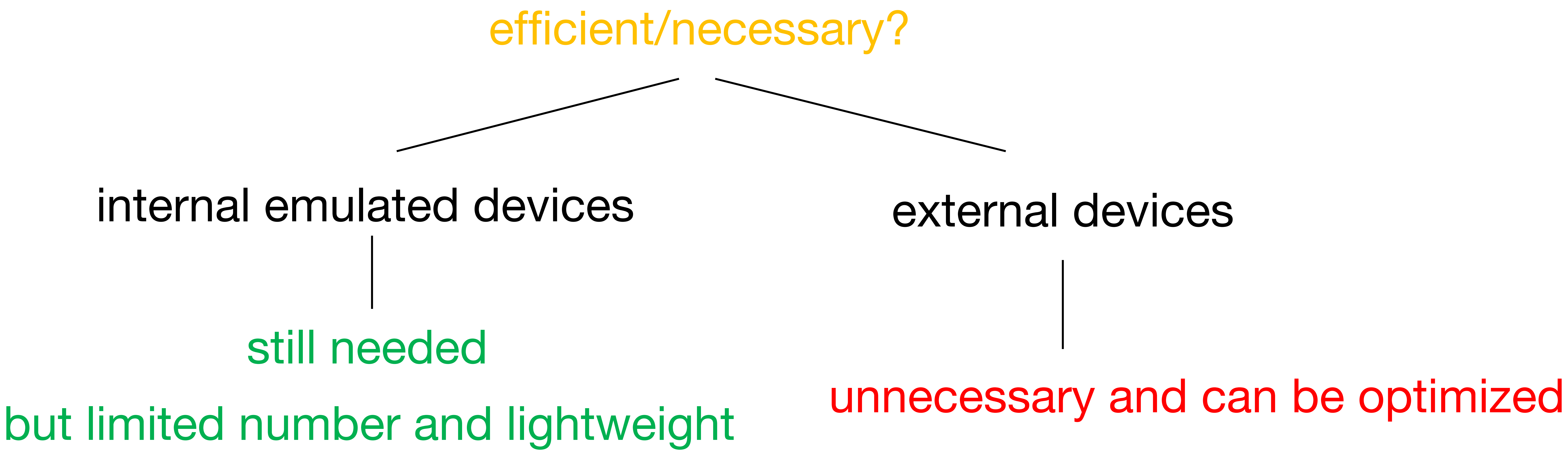
Main time cost:

- stop/start vhost-uesr devices
- transfer device state

Optimizations

Optimizations - Insight

Directly reuse the live migration framework to stop/start devices and transfer device state



efficient/necessary?

internal emulated devices

still needed

but limited number and lightweight

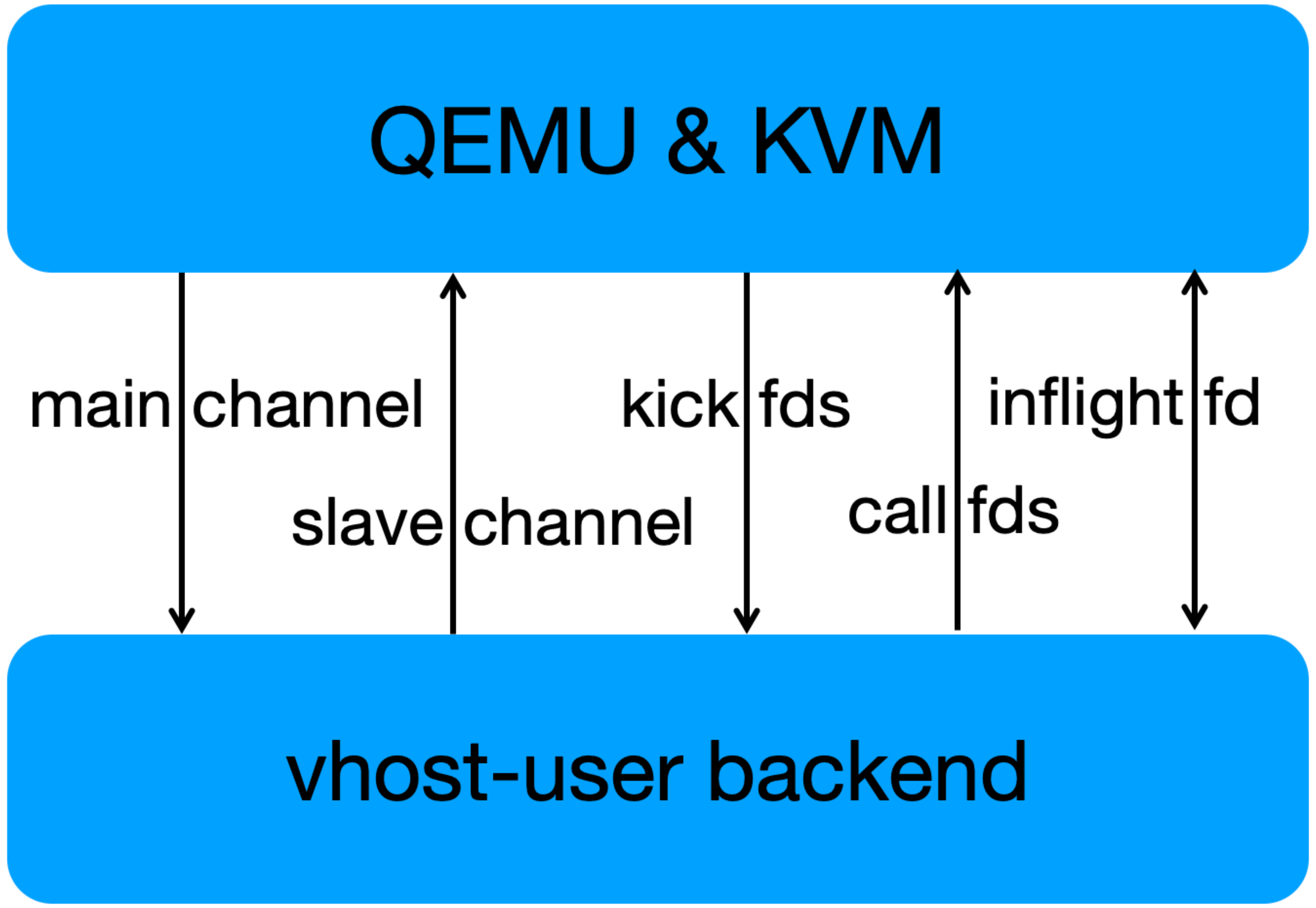
external devices

unnecessary and can be optimized

Optimizations - Transparent to Backends

Take vhost-user devices (DPDK/SPDK) for example

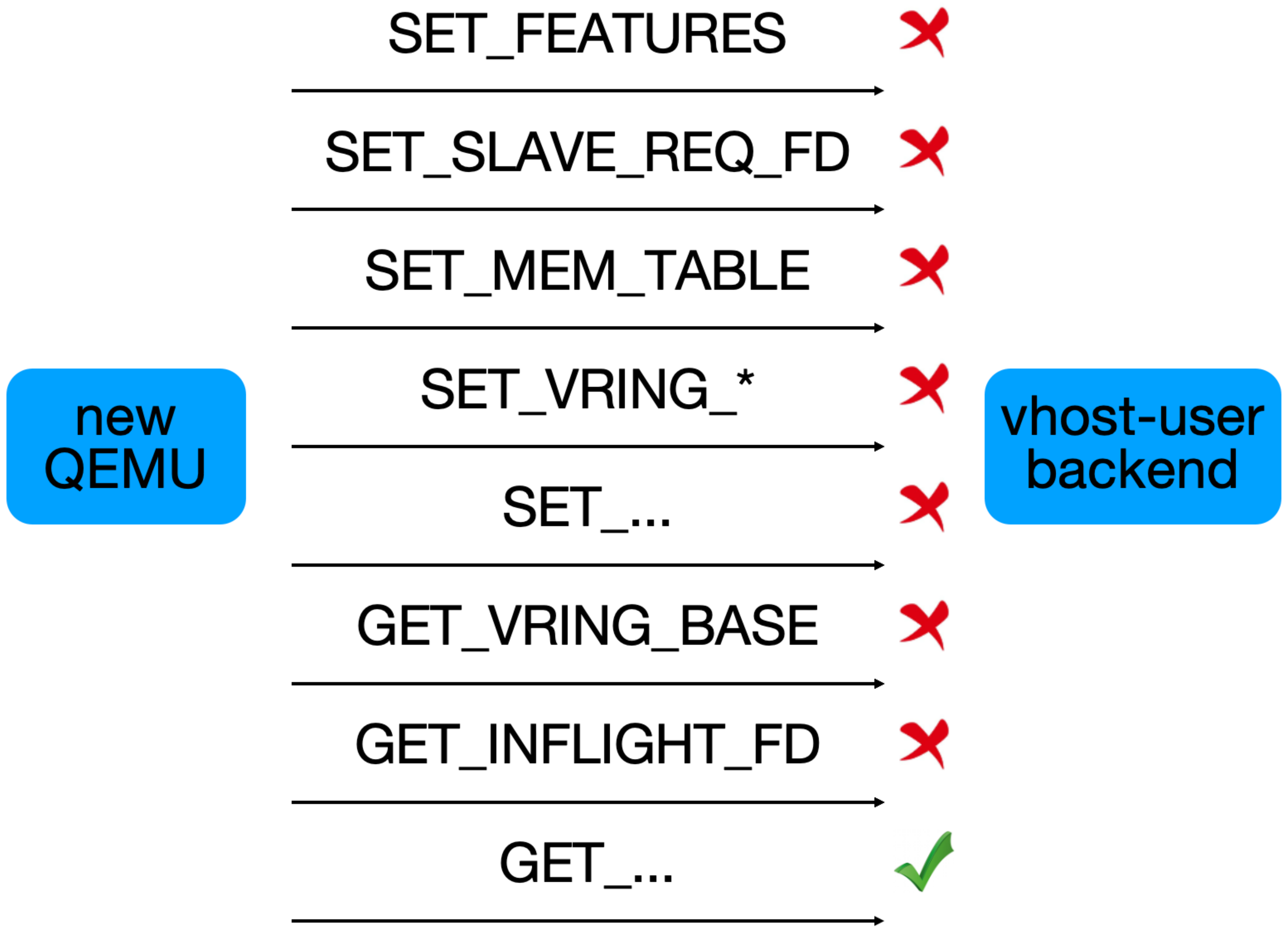
- Inherit the channels and shared fds between the VMM and vhost-user backends
- Use them directly in the new QEMU and skip the related init processes



Optimizations - Transparent to Backends

Make vhost-user backends unaware of in live upgrade:

- don’t stop the backends in the old QEMU
- skip all ‘set’ and some ‘get’ communications to the backends in the new



Optimizations - Transparent to Backends

Some issues:

- the backends keep running and may trigger IRQs even after the guest has paused, then the new kvm may miss the IRQs received and pending in the old

  simply supplement IRQs unconditionally when finishing the upgrade

- if the backends crash or send SLAVE_* messages to the master, it is uncertain which QEMU will receive the messages…

  the new QEMU start to listen on the slave channel only when finishing the upgrade, and if there is any backend crash or slave request, just fail this upgrade

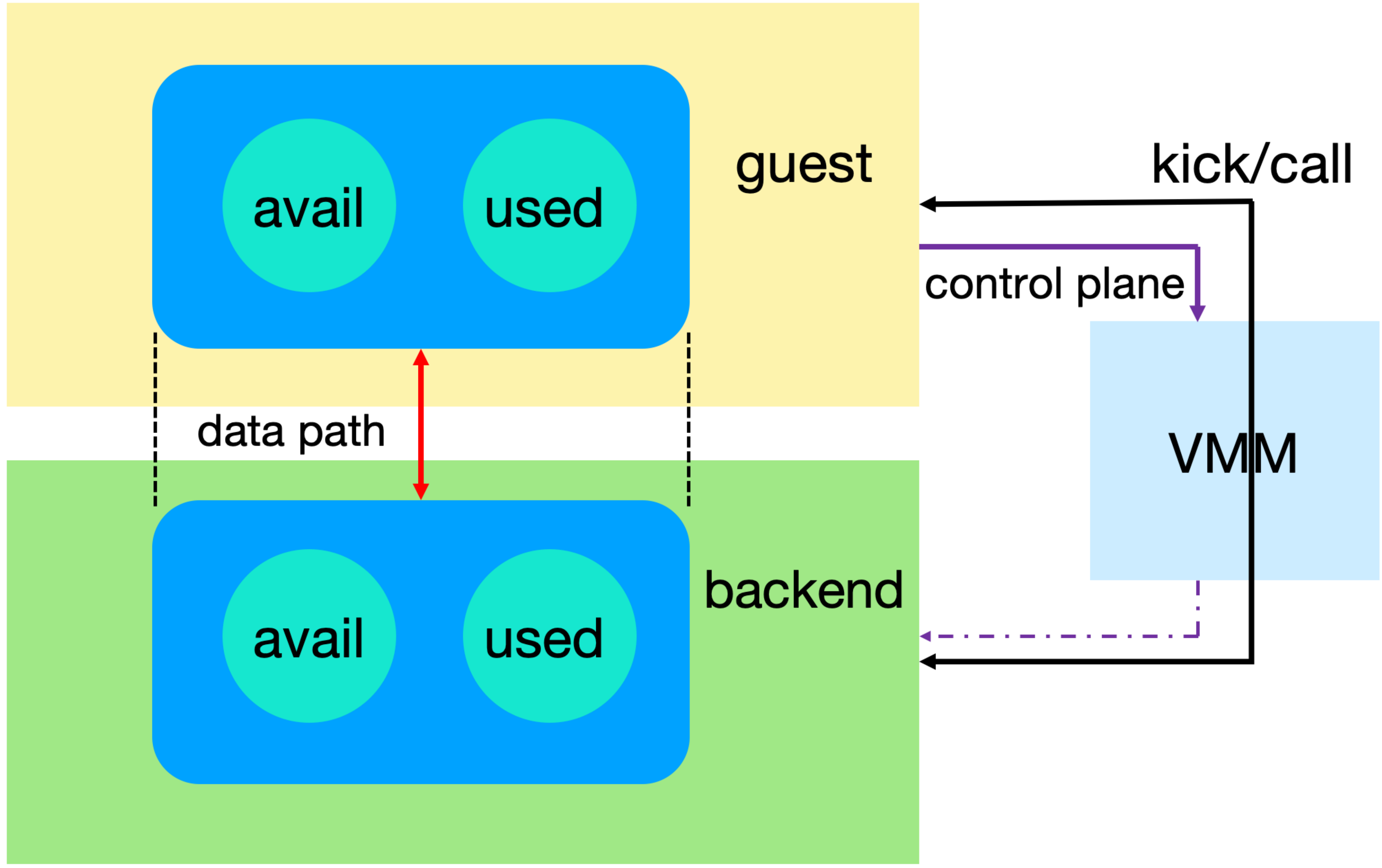
- cause stale mem-table data in the backends

  merely update the data or mmap the guest RAM at a fixed and very high address

Optimizations - Presave Config

- Virtqueue-related state in the data plane is kept in the guest and backends
  - no need to transfer
- Config state is much less changed during the VM lifetime
  - presave it before VM pause

  keep a track of the config change, and retransfer the state after VM pause if any change occurs



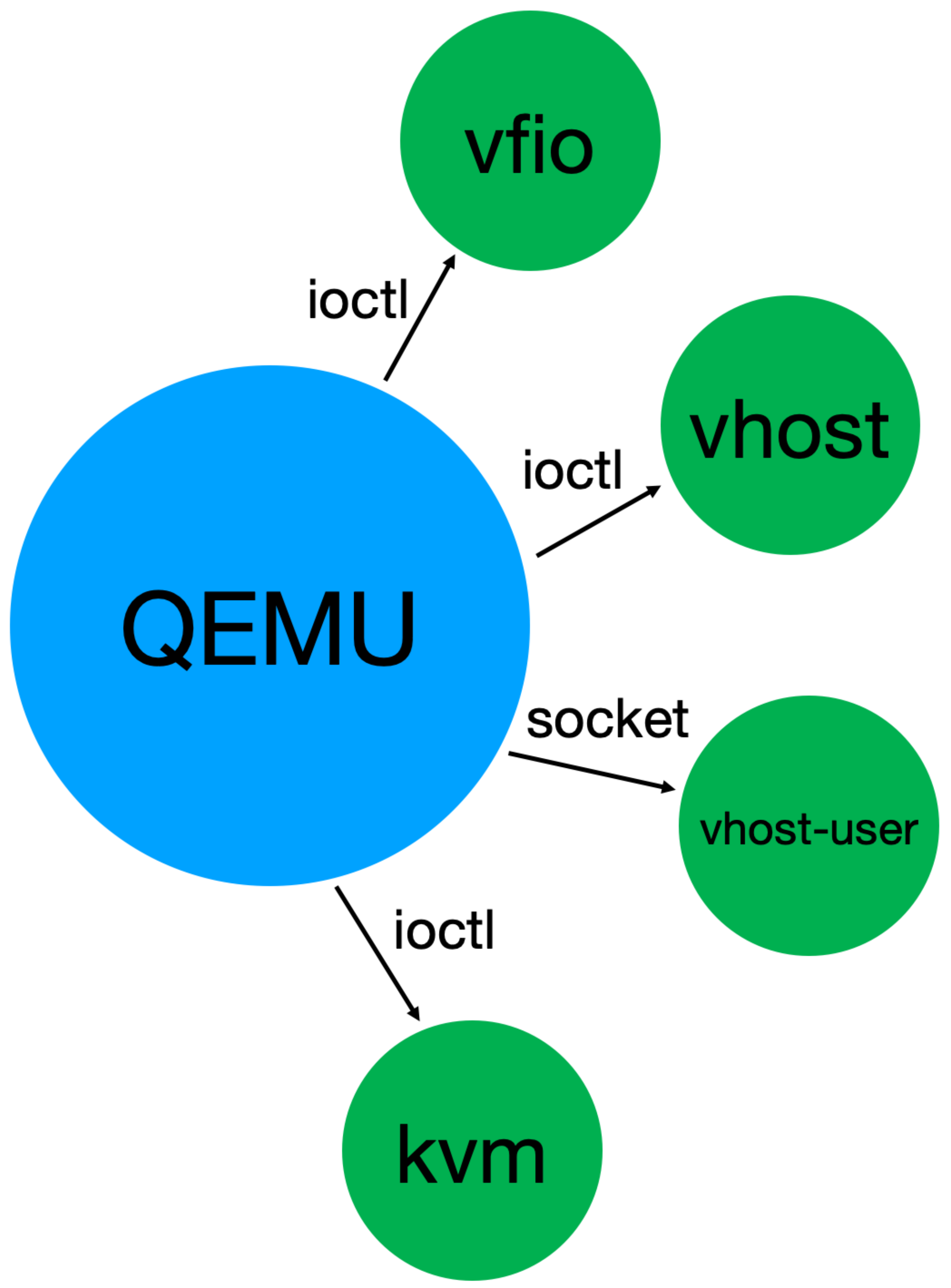
Optimizations - More Than Vhost-user

- Also apply to vfio, vhost…

- QEMU upgrade only mode

  - inherit the kvm fds and skip the related init processes

  - no need to sync the vCPU state from/to kvm

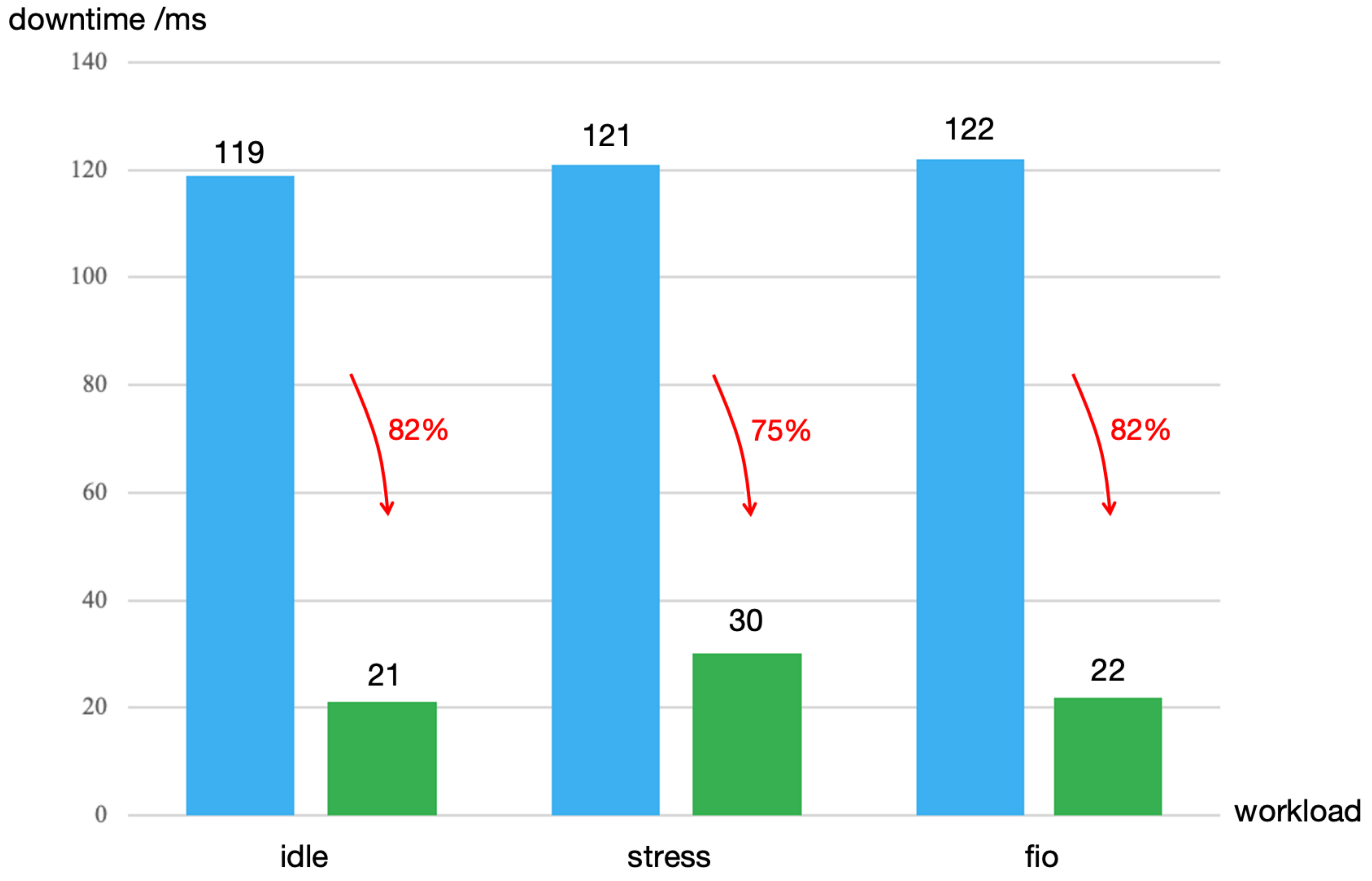


Achievements

ByteDance

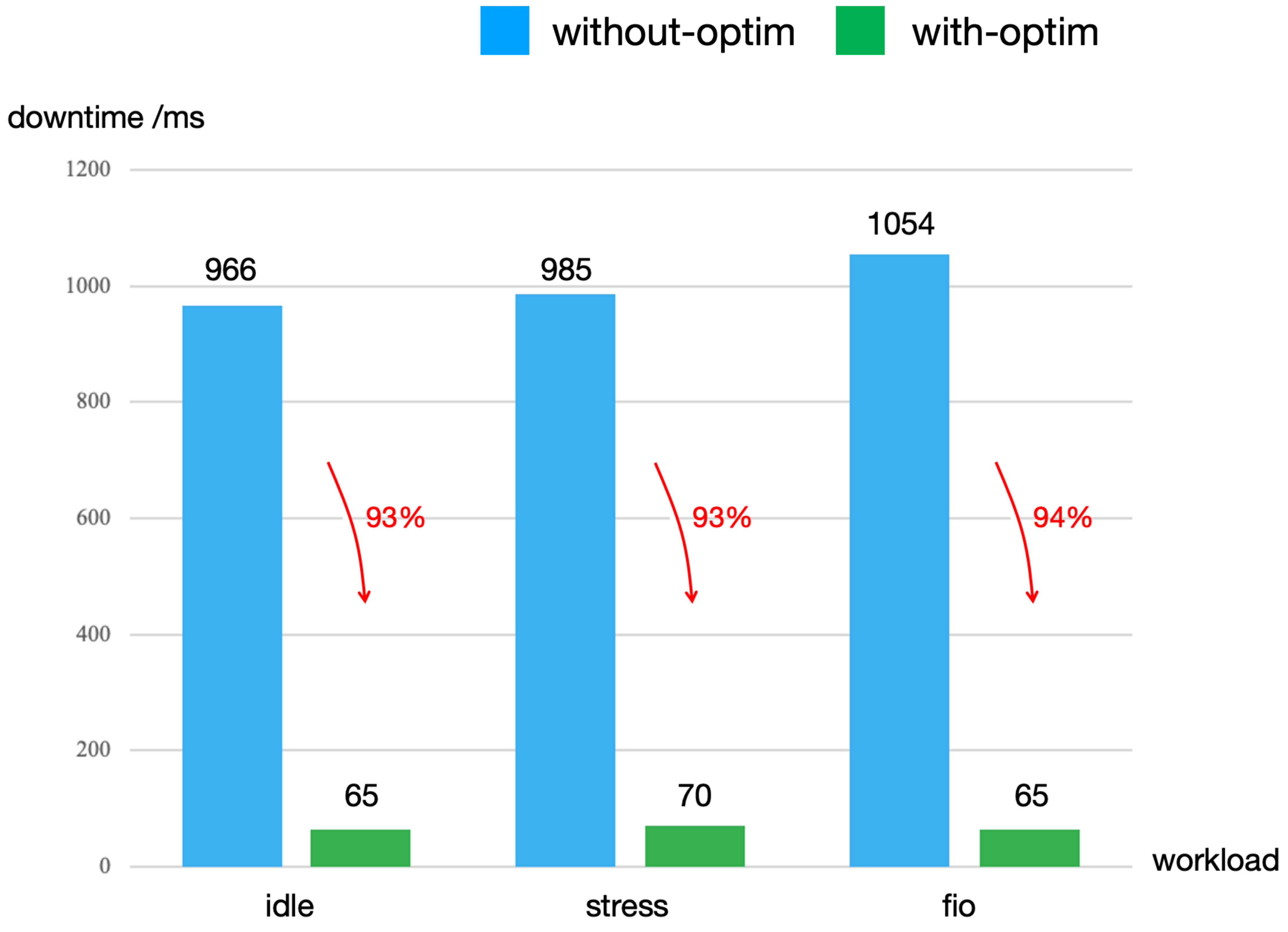
Achievements - Downtime

- Effect of optimizations on downtime



stress: stress -c 16
fio: fio -filename=/dev/vdb -rw=randwrite -direct=1 -bs=4k -iodepth=4 …

-1-



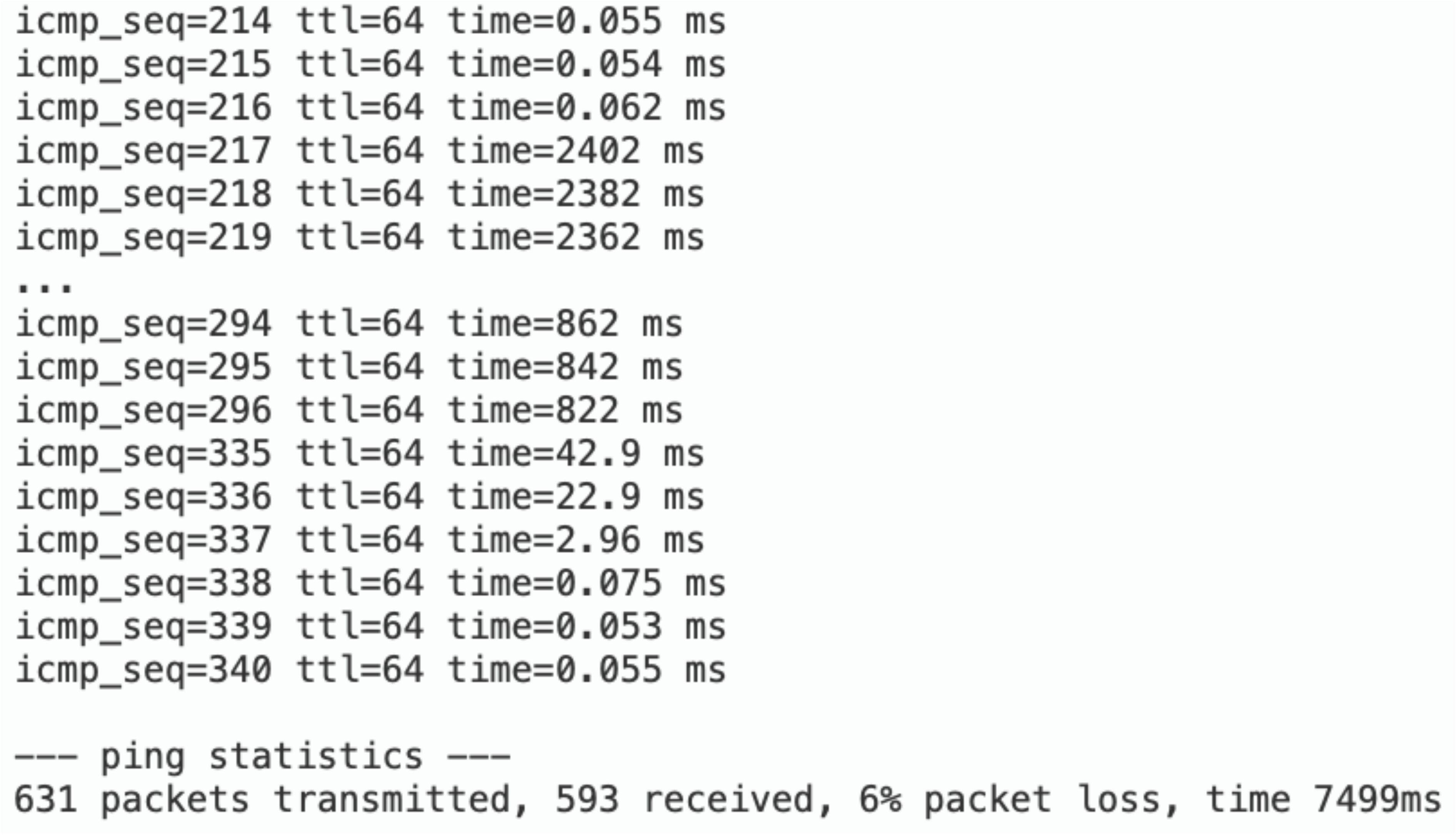
stress: stress -c 64
fio: fio -filename=/dev/vdb:/dev/vdc… -rw=randwrite -direct=1 -bs=4k -iodepth=4 …

-2-

-1- 16 vCPUs, 64G memory, 1 multiqueue vhost-user-net devices, 2 multiqueue vhost-user-blk devices
-2- 64 vCPUs, 256G memory, 2 multiqueue vhost-user-net devices, 10 multiqueue vhost-user-blk devices

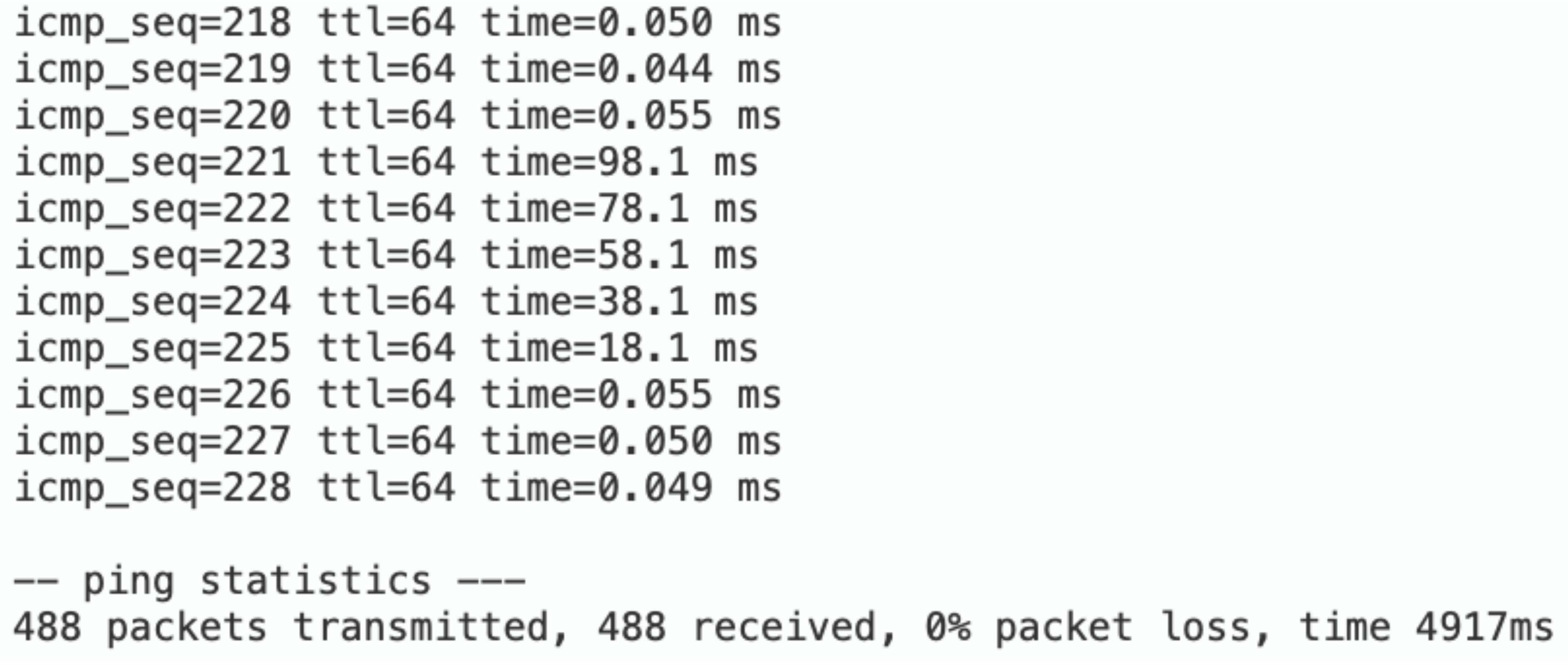
Achievements - Packet Loss

- Effect of optimizations on packet loss



```
icmp_seq=214 ttl=64 time=0.055 ms
icmp_seq=215 ttl=64 time=0.054 ms
icmp_seq=216 ttl=64 time=0.062 ms
icmp_seq=217 ttl=64 time=2402 ms
icmp_seq=218 ttl=64 time=2382 ms
icmp_seq=219 ttl=64 time=2362 ms
...
icmp_seq=294 ttl=64 time=862 ms
icmp_seq=295 ttl=64 time=842 ms
icmp_seq=296 ttl=64 time=822 ms
icmp_seq=335 ttl=64 time=42.9 ms
icmp_seq=336 ttl=64 time=22.9 ms
icmp_seq=337 ttl=64 time=2.96 ms
icmp_seq=338 ttl=64 time=0.075 ms
icmp_seq=339 ttl=64 time=0.053 ms
icmp_seq=340 ttl=64 time=0.055 ms

--- ping statistics ---
631 packets transmitted, 593 received, 6% packet loss, time 7499ms
```



```
icmp_seq=218 ttl=64 time=0.050 ms
icmp_seq=219 ttl=64 time=0.044 ms
icmp_seq=220 ttl=64 time=0.055 ms
icmp_seq=221 ttl=64 time=98.1 ms
icmp_seq=222 ttl=64 time=78.1 ms
icmp_seq=223 ttl=64 time=58.1 ms
icmp_seq=224 ttl=64 time=38.1 ms
icmp_seq=225 ttl=64 time=18.1 ms
icmp_seq=226 ttl=64 time=0.055 ms
icmp_seq=227 ttl=64 time=0.050 ms
icmp_seq=228 ttl=64 time=0.049 ms

-- ping statistics ---
488 packets transmitted, 488 received, 0% packet loss, time 4917ms
```

without-optim

with-optim

64 vCPUs, 256G memory, 2 multiqueue vhost-user-net devices, 10 multiqueue vhost-user-blk devices

much lower latency and no packet loss

Thank You

ByteDance

Q & A

Contact Info： lushenming@bytedance.com

ByteDance